An efficient spectral interpolation routine for the TwoPunctures code

Vasileios Paschalidis,[1] Zachariah B. Etienne,[1] Roman Gold,[1] and Stuart L. Shapiro[1,*]

[1] *Department of Physics, University of Illinois at Urbana-Champaign, Urbana, IL 61801*

TwoPunctures is perhaps the most widely-adopted code for generating binary black hole "puncture" initial data and interpolating these (spectral) data onto evolution grids. In typical usage, the bulk of this code's run time is spent in its spectral interpolation routine. We announce a new publicly-available spectral interpolation routine that improves the performance of the original interpolation routine by a factor of ∼100, yielding results consistent with the original spectral interpolation routine to roundoff precision. This note serves as a guide for installing this routine both in the original standalone TwoPunctures code and the Einstein Toolkit supported version of this code.

PACS numbers: 04.25.dg, 04.40.Nr, 04.25.D-, 04.25.dk

arXiv:1304.0457v2 [gr-qc] 3 Apr 2013

I. INTRODUCTION

One of the pressing goals of numerical relativity is to calculate accurate gravitational waveforms from plausible astrophysical sources to help generate templates that will be used by ground based gravitational wave observatories such as LIGO [1, 2], VIRGO [3, 4], TAMA [5, 6],GEO [7], KAGRA [8], and by proposed space-based interferometers such as eLISA/NGO [9] and DECIGO [10]. This task is far from trivial and the very first step in accomplishing it is the generation of initial data that satisfy the Einstein constraints [11].

To date there are multiple codes that solve the constraints of Einstein's theory of general relativity (see e.g. [12–17] and references therein). As the inspiral and merger of compact binaries such as binary black holes (BHBH), binary neutron stars and binary black hole–neutron stars, furnish some of the most promising astrophysical scenarios (both in terms of signal strength and event rates) for the generation of detectable gravitational waves, the solutions obtained using these initial data solvers are mainly focused on compact binary systems.

Most of the recent focus in gravitational wave template generation has been on binary black hole systems (see e.g. [18, 19]) and for this reason one of the most popular codes for generating BHBH initial data is the TwoPunctures code [17], which has also been adopted by the publicly available Einstein Toolkit [20]. The popularity of this code stems from the fact that it is remarkably user-friendly, spectrally accurate and efficient in solving the Einstein constraints for BHBHs when both BHs are represented as punctures. Once the initial value problem has been solved, the initial data have to be mapped onto the dynamical evolution grids via interpolation. The TwoPunctures code offers two interpolation routines: i) a second-order polynomial interpolation routine, and ii) a spectral interpolation routine. The former is very fast but we have found empirically that it is not well-suited for dynamical evolutions. For this reason, it is the latter that is most widely used by us and most numerical relativity groups.

In this brief note we provide a spectral interpolation routine for the TwoPunctures code (and installation instructions both for the Einstein Toolkit version of the code, i.e., the TwoPunctures thorn, and its standalone version) that is ~ 100 times faster than the original spectral interpolation routine of the TwoPunctures code. This new routine saves many hours of computation, especially when high spectral resolutions are used. By no means do we claim that we have optimized the process, and one may find other ways of optimizing the performance of the code in general (see below). However, we find that the acceleration attained by our routine is sufficiently satisfactory, and we hope that by making this routine publicly available, the many numerical relativity groups that use TwoPunctures will benefit from this faster spectral interpolation routine.

II. A FASTER SPECTRAL INTERPOLATION ROUTINE

Our basic modification of the TwoPunctures code was stimulated by the observation that each time the original TwoPunctures code spectral interpolation routine is called it computes the spectral interpolation coefficients, given the values of the function at the collocation points, and then uses the spectral expansion to interpolate to any one Cartesian grid point. Typical high resolution evolution grids in finite difference codes employ 8-9 levels of refinement with resolutions of $M/40$ or higher, where M is the BHBH ADM mass. This amounts to a grid of about 10^5-10^6 zones

* Also at Department of Astronomy and NCSA, University of Illinois at Urbana-Champaign, Urbana, IL 61801

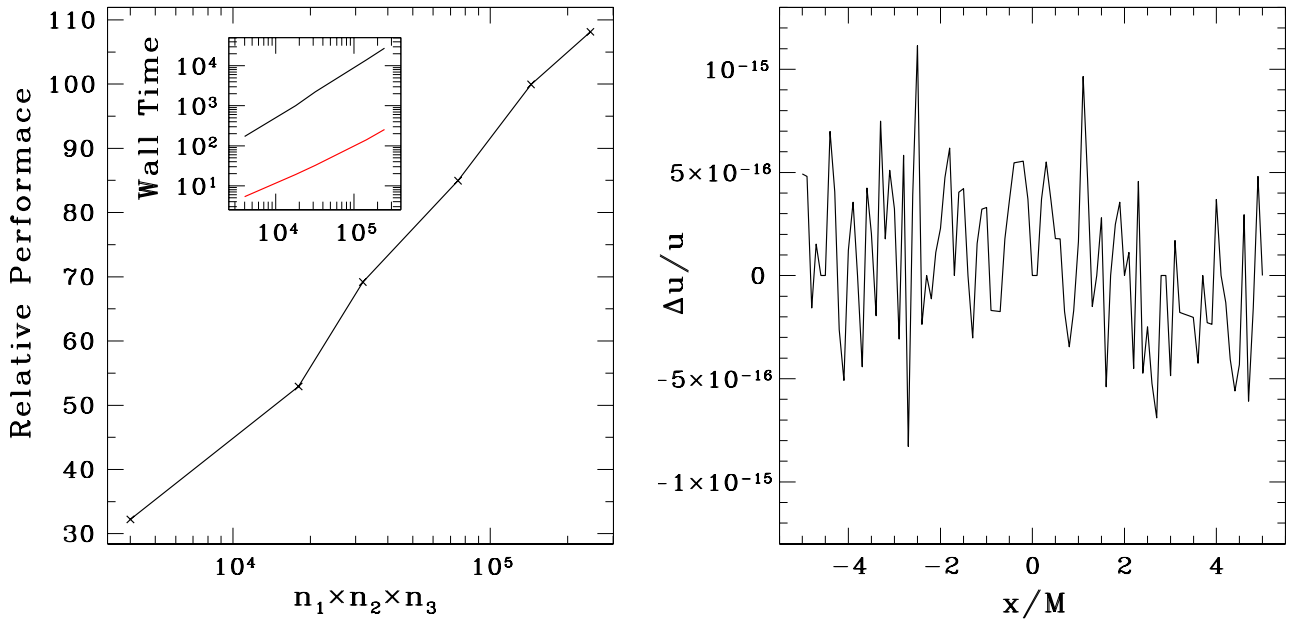

FIG. 1. Left:Performance of the original `TwoPunctures` code interpolation routine normalized by the performance of our new spectral interpolation routine versus the total number of basis functions (or collocation points). Performance is measured in seconds of wall time. The inset shows the required wall time in seconds. The black curve corresponds to the original `TwoPunctures` code interpolation routine and the red curve to our spectral interpolation routine. The performance test is based on timing the interpolation operation for a grid consisting of 10^5 zones, after the code has solved for a particular BHBH configuration. The spectral resolutions used were $(n_1, n_2, n_3) = (10, 20, 20)$, $(20, 30, 30)$, $(20, 40, 40)$, $(30, 50, 50)$, $(40, 60, 60)$, $(50, 70, 70)$. Right: Fractional difference between the interpolated values along the BHBH binary axis (the punctures lie at $x/M = \pm 2.17$) using the original `TwoPuncture` code interpolation routine and our new spectral interpolation routine. The results agree to machine precision. All runs were performed on a system with an Intel Core 2 Duo 6300 processor. The code was compiled with the Intel 11.1 C++ compiler with -O3 optimizations. Similar relative performance is found on other systems and with other compiler optimizations.

for which interpolations must be performed. Calculating the spectral coefficients from the values at the collocation points is an expensive operation, which is exacerbated by computing them at every interpolated point. In our initial value calculations we use the `TwoPunctures` code with 50×70^2 basis functions, thus the original `TwoPunctures` code spectral interpolation routine calculates $\sim 2.5 \times 10^5$ spectral coefficients each time it is called, i.e., $\sim 10^6$ times. This means that the spectral coefficients are recomputed $\sim 10^{11}$ times in this process.

The total cost of this operation can be reduced significantly, if the spectral coefficients computation is a one-time operation. So, our new interpolation routine uses the Chebyshev and Fourier basis routines in the `TwoPunctures` code to compute the spectral coefficients once and for all, and then store them in an array. Another routine we developed takes the stored spectral coefficients as input and performs the spectral interpolation. To compute the required sums we use the partial summation method [21]. Our new routine still computes the values of the bases functions at the collocation points in order to perform the interpolation every time it is called, hence further optimization can take place by storing the values of the bases functions at the collocation points as well. However, we did not do so because the new interpolation routine performs sufficiently fast.

In the left panel of Fig. 1 we show the performance of the original `TwoPunctures` spectral interpolation routine normalized by the performance of our routine against the total number of collocation points for a test case where we do 10^5 interpolations after solving for an initial BHBH configuration. It is clear that our routine accelerates the interpolation procedure. The higher the spectral resolution the greater the gain, and for resolutions of 50×70^2 we get more than a factor of 100 speed-up: our routine takes ~ 200s, while the original routine requires ~ 28000s to finish. We have also checked that the results with the two routines agree to machine precision as shown in the right panel of Fig. 1.

III. INSTRUCTIONS FOR INSTALLING THE NEW INTERPOLATION ROUTINE

Here we provide instructions for installing the new interpolation routine both in the `TwoPunctures` thorn of the Einstein Toolkit, and the standalone version of the `TwoPunctures` code. The routines for the Einstein Toolkit version of the `TwoPunctures` code can be downloaded from http://webusers.physics.illinois.edu/~vpaschal/TwoPuncturesET/. The routines for the standalone `TwoPunctures` code can be downloaded from http://webusers.physics.illinois.edu/~vpaschal/TwoPunctures_Standalone/.

A. `TwoPunctures` Einstein Toolkit

We provide the routines

```
void SpecCoef(int n1, int n2, int n3, int ivar, CCTK_REAL *v, CCTK_REAL *cf)

CCTK_REAL PunctIntPolAtArbitPositionFast(int ivar, int nvar, int n1,
                                         int n2, int n3, derivs v,
                                         CCTK_REAL x, CCTK_REAL y, CCTK_REAL z)

CCTK_REAL PunctEvalAtArbitPositionFast(CCTK_REAL *v, int ivar, CCTK_REAL A,
                                       CCTK_REAL B, CCTK_REAL phi, int nvar,
                                       int n1, int n2, int n3)
```

The first computes the spectral expansion coefficients (`cf`) of a variable (`v`) given the values of variable `v` at the collocation points, where `n1, n2, n3`, are the number of basis functions in the A, B, ϕ coordinates of the `TwoPunctures` code. The second routine applies spectral interpolation on the variable `v` to the desired point with Cartesian coordinates x, y, z, using the third routine which interpolates `v` using the corresponding `TwoPunctures` coordinates A, B, ϕ. The following are the required steps to implement our new interpolation routine.

1. For convenience we recommend that these routines be added in the file FuncAndJacobian.c of the `TwoPunctures` thorn.

2. These routines must also be declared in the TwoPunctures.h header file.

3. In the file TwoPuncture.c the following declaration statement must be added:

   ```
   static derivs cf_v;
   ```

4. In the file TwoPuncture.c the following memory allocation call for storage of the spectral coefficients must be added:

   ```
   allocate_derivs (&cf_v, ntotal);
   ```

5. Following memory allocation, the new variables should be initialized (e.g.):

   ```
   for (int j = 0; j < ntotal; j++)
   {
     cf_v.d0[j] = 0.0;  cf_v.d1[j] = 0.0;
     cf_v.d2[j] = 0.0;  cf_v.d3[j] = 0.0;
     cf_v.d11[j] = 0.0; cf_v.d12[j] = 0.0;
     cf_v.d13[j] = 0.0; cf_v.d22[j] = 0.0;
     cf_v.d23[j] = 0.0; cf_v.d33[j] = 0.0;
   }
   ```

   Note that there is a redundancy, as `cf_v.d0` is the only required variable, but as the memory footprint of the `cf_v` derivs struct is small this redundancy is of minor significance.

6. In the file TwoPuncture.c following the function call

```
F_of_v (cctkGH, nvar, n1, n2, n3, v, F, u);
```

which follows the loop checking for convergence of the puncture masses, the call to the calculation of the spectral coefficients should be made

```
SpecCoef(n1, n2, n3, 0, v.d0, cf_v.d0);
```

7. All calls to `PunctIntPolAtArbitPosition` should be replaced by corresponding calls to `PunctIntPolAtArbitPositionFast`, but the most important change is to replace the call

```
U = PunctIntPolAtArbitPosition(0, nvar, n1, n2, n3, v, x1, y1, z1);
```

by the call

```
U = PunctIntPolAtArbitPositionFast(0, nvar, n1, n2, n3, cf_v, x1, y1, z1);
```

8. Memory for the spectral coefficients should be freed at the end of the TwoPuncture.c file

```
free_derivs (&cf_v, ntotal);
```

B. Standalone `TwoPunctures` code

For the standalone `TwoPunctures` code we provide the following routines:

```
void SpecCoef(parameters par, int ivar, double *v, double *cf)
double Spec_IntPolFast (parameters par, int ivar, double *v, double x, double y, double z)
double Spec_IntPolABphiFast (parameters par, double *v, int ivar, double A, double B, double phi)
```

These routines do precisely the same calculations as the routines `SpecCoef`, `PunctIntPolAtArbitPositionFast`, `PunctEvalAtArbitPositionFast` in the `TwoPuncturesET` version of the code.

The steps to implement our new interpolation routine in the standalone version are similar to the ones we outlined in the previous section III A

1. These routines can be added to the file FuncAndJacobian.C of the `TwoPunctures` code.
2. These routines must be declared in the TwoPunctures.h header file.
3. In the file TwoPunctures.C the following declaration statement must be added:

```
derivs cf_v;
```

4. In the file TwoPunctures.C memory should be allocated for storage of the spectral coefficients:

```
allocate_derivs (&cf_v, ntotal);
```

5. In the file TwoPunctures.C following the call

```
Newton(par, v);
```

the calculation of the spectral coefficients should be performed:

```
SpecCoef(n1, n2, n3, 0, v.d0, cf_v.d0);
```

6. Calls to

```
Spec_IntPol(par, 0, v.d0, x, y, z);
```

should be replaced by calls to

```
Spec_IntPolFast(par, 0, cf_v.d0, x, y, z);
```

7. Memory for the spectral coefficients should be freed at the end of the TwoPunctures.C file

```
free_derivs (&cf_v, ntotal);
```

ACKNOWLEDGMENTS

This work was supported in part by NSF Grants PHY-0963136 as well as NASA Grant NNX11AE11G to the University of Illinois at Urbana-Champaign. Z. Etienne gratefully acknowledges support from NSF Astronomy and Astrophysics Postodoctoral Fellowship AST-1002667.

[1] B. Abbott and the LIGO Scientific Collaboration, Phys. Rev. D **77**, 062002 (Mar. 2008)
[2] D. A. Brown, S. Babak, P. R. Brady, N. Christensen, T. Cokelaer, J. D. E. Creighton, S. Fairhurst, G. Gonzalez, E. Messaritaki, B. S. Sathyaprakash, P. Shawhan, and N. Zotov, Class. Quant. Grav. **21**, S1625 (Oct. 2004)
[3] F. Acernese and the VIRGO Collaboration, Class. Quant. Grav. **23**, S635 (Oct. 2006)
[4] F. Beauville and the LIGO-VIRGO Working Group, Classical and Quantum Gravity **25**, 045001 (Feb. 2008)
[5] M. Ando and the TAMA collaboration, Class. Quant. Grav. **19**, 1409 (Apr. 2002)
[6] D. Tatsumi and the TAMA collaboration, Classical and Quantum Gravity **24**, S399 (Oct. 2007)
[7] H. Lück and the GEO600 collaboration, Class. Quant. Grav. **23**, S71 (Apr. 2006)
[8] K. Kuroda and LCGT Collaboration, Classical and Quantum Gravity **27**, 084004 (Apr. 2010)
[9] P. Amaro-Seoane, S. Aoudia, S. Babak, P. Binetruy, E. Berti, *et al.*(2012), arXiv:1201.3621 [astro-ph.CO]
[10] S. Kawamura and the DECIGO collaboration, Class. Quant. Grav. **23**, S125 (Apr. 2006)
[11] T. W. L. Baumgarte and S. L. Shapiro, *Numerical Relativity* (Cambridge University Press, 2010)
[12] H. P. Pfeiffer, L. E. Kidder, M. A. Scheel, and S. A. Teukolsky, Comput.Phys.Commun. **152**, 253 (2003), arXiv:gr-qc/0202096 [gr-qc]
[13] `http://www.lorene.obspm.fr/`
[14] V. Paschalidis, Z. Etienne, Y. T. Liu, and S. L. Shapiro, Phys.Rev. **D83**, 064002 (2011), arXiv:1009.4932 [astro-ph.HE]
[15] V. Paschalidis, Y. T. Liu, Z. Etienne, and S. L. Shapiro, Phys.Rev. **D84**, 104032 (2011), arXiv:1109.5177 [astro-ph.HE]
[16] W. E. East, F. M. Ramazanoglu, and F. Pretorius, Phys.Rev. **D86**, 104053 (2012), arXiv:1208.3473 [gr-qc]
[17] M. Ansorg, B. Bruegmann, and W. Tichy, Phys.Rev. **D70**, 064011 (2004), arXiv:gr-qc/0404056 [gr-qc]
[18] P. Ajith, S. Babak, Y. Chen, M. Hewitson, B. Krishnan, A. M. Sintes, J. T. Whelan, B. Brügmann, P. Diener, N. Dorband, J. Gonzalez, M. Hannam, S. Husa, D. Pollney, L. Rezzolla, L. Santamaría, U. Sperhake, and J. Thornburg, Phys. Rev. D **77**, 104017 (May 2008)
[19] P. Ajith, M. Boyle, D. A. Brown, B. Brugmann, L. T. Buchman, *et al.*, Class.Quant.Grav. **29**, 124001 (2012), arXiv:1201.5319 [gr-qc]
[20] F. Loffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, *et al.*, Class.Quant.Grav. **29**, 115001 (2012), arXiv:1111.3344 [gr-qc]
[21] J. P. Boyd, *Chebyshev and Fourier Spectral Methods: Second Revised Edition* (Dover Books on Mathematics, 2001)